Enabling Collaboration in Virtual Reality Navigators
Theoktisto, V. and Fairén, M. and Navazo, I.
Research Report LSI-04-13-R

Departament de Llenguatges i Sistemes Informàtics

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Enabling collaboration in Virtual Reality Navigators

Víctor Theoktisto
Marta Fairén
Isabel Navazo

Report LSI-04-13-R

Enabling Collaboration in Virtual Reality Navigators

Victor Theoktisto Marta Fairén Isabel Navazo

Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract

In this paper we characterize a feature superset for Collaborative Virtual Reality Environments (CVRE), and derive a component framework to transform stand-alone VR navigators into full-fledged multithreaded collaborative environments. The contributions of our approach rely on a cost-effective and extensible technique for loading software components into separate POSIX threads for rendering, user interaction and network communications, and adding a top layer for managing session collaboration. The framework recasts a VR navigator under a distributed peer-to-peer topology for scene and object sharing, using callback hooks for broadcasting remote events and multicamera perspective sharing with avatar interaction. We validate the framework by applying it to our own ALICE VR Navigator. Experimental results show that our approach has good performance in the collaborative inspection of complex models.

Categories and Subject Descriptors (according to ACM CCS):
I.3.7 [Computer Graphics]: Virtual Reality; H.5.3 [Group and Organization Interfaces]: Collaborative Computing

1. Introduction

Virtual Reality (VR) techniques have been applied in many engineering fields as powerful tools to avoid or reduce the use of physical prototypes, to recreate emergency situations, to train in dangerous locales, and to interpret real or simulated results. In medical applications VR helps patient monitoring, interpretation of scanned data, and surgery planning. In architectural settings VR allows designing, building, visiting and stress-testing upcoming facilities. In these virtual reality environments (VREs), individual users inspect 3D scenes, navigate inside models and manipulate objects and their properties. Most implementations of VREs have begun as stand-alone applications, with collaboration requests surging as soon as the users decide to repeat or interchange experiences. Allowing for several users to collaborate on the inspection of a model usually requires the development of a whole new application with distributed capabilities, using custom or available network communication libraries, and in general confronting code portability problems.

We propose a framework for naturally evolving full-fledged Collaborative Virtual Reality Environments (CVRE) out of stand-alone VR applications, provided that the original application is well designed and has separate rendering and user interface components. The contributions of our approach incorporate a distributed user interaction model, multi-threaded software components, network communications under a scalable peer-to-peer topology, message passing channels, and several user roles in a multicamera subscription model. The framework adjusts to the existing functioning code base with minimal tinkering, and does not adversely affect performance.

In section 2 we review relevant previous work, collect a feature superset of CVREs under several characterizations, and explain current user interaction paradigms.

We develop in section 3 the generic blueprint for the transformation of suitable VR navigators into small or medium scale CVREs, in the form of a snap-on minimal framework providing network and session management.

In section 4 we introduce the ALICE VR Real Time Inspector and Navigator, a stand-alone VR Navigator developed at the Universitat Politècnica de Catalunya, and validate the developed framework by transforming ALICE in a complete collaborative environment.

Section 5 shows performance results under the framework, testing several sets of enhanced ALICE clients in a busy network, each fitted with output displays such as Head-Mounted Displays, a portable VR system, stereoscopic tables and other monitors.

Finally in section 6 we plan for including new capabilities in the framework, such extension to other VR tools, adding a new component allowing faster channel for haptic devices, and multi-resolution streaming techniques for transmitting large models across participants.

2. Collaborative Virtual Reality Environments

Distributed environments have been around since the introduction of the first networks. Scope and complexity have kept pace with distributed systems evolution, migrating towards distributed processing, data sharing, multiple execution threads and sophisticated display technology. Computer Support for Cooperative Work(CSCW) [CSS99] is an umbrella term for distributed applications in which multiple users collaborate toward common goals, under a high level event notification and message passing architecture. When combined with several degrees of information sharing and screen visualization, they are known as Collaborative Virtual Environments or CVE's.

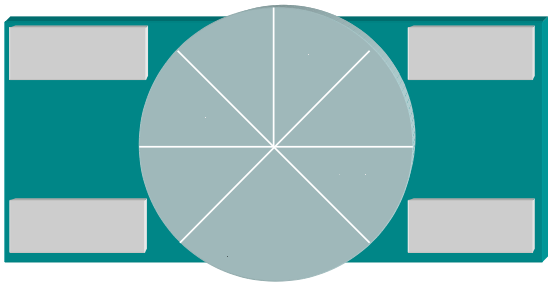


Figure 1: *Components of Collaborative Virtual Reality Environments. The items placed at the upper half of the circle are those directly related to collaboration issues.*

When a simulated 3D universe is implemented as a CVE, with users taking visual identities (called *avatars*) inside the environment, it is then called a Collaborative Virtual Reality Environment (CVRE). Avatars can navigate around the 3D world, collaborate with other avatars in real time, and may propagate changes to objects in the environment. Environment and object manipulation require sophisticated user interaction models and interfaces. Figure 1 details all the software components which are present in a complete CVRE.

The first developed major distributed virtual environments were DIVE [CH93] and MASSIVE [GB94]. A review of simulation CVE's is presented in [Sty96], addressing issues in human-computer interaction, Virtual Reality, software engineering, 3D graphics, scene modeling/object modeling, and artificial intelligence. Duce *et al.* [DGC*98] characterize reference models for CVEs and the impact they have on the degree of collaboration. A different approach is presented in [MF98], where the CVE is included into a library called REPO-3D, which allows the migration/replication of graphic objects over the network.

Different network technologies can be used to enable distribution on a CVE (BSD-sockets, RPC, Java RMI, DCOM, CORBA, etc.). Avocado [Tra99], DIVE, NAVL [WLG99], NPSNET-V [CMBZ00], and Distributed Open Inventor [HSFP99] apply different solutions, using multiple execution threads where each one has an image of the other participants in the interaction. Each client requires, thus, a partial image of the scene graph. Zeleznik *et al.* [ZHC*00] use the scene graph as a communications bus instead of a tree, whose nodes are sited at different network nodes and are accessed by synchronized access mechanisms. Diverse [KSA*03] uses remote shared memory and UDP network datagrams for a rapid memory interchange. Treatment of temporal inconsistencies due to network delays are detailed in [Mau00] and [MW01], while [GFPB02] describes a technique for embedding temporal links in a CVRE.

The most sophisticated approaches delegate clients' network management to components outside of the CVE. Some of them use customized solutions, like Octopus and Tweek for VRJuggler [HJCN01] or CAVERNSoft for CAVELib [LJD96]. Other are based on the use of the CORBA standard [DKS*99] for object sharing over the network, having a central object registry and a localization service.

2.1. Characterization of Collaboration in Virtual Reality Environments

Treatment of remote collaboration capabilities in a CVRE can be characterized by the categories summarized in Table 1. We have grouped together those features that apply to generic CVEs (the first three) and highlighted those specific to CVREs (the last two). A more detailed exposition and relevant references can be found in [TFN03].

Features include: *network substrate* to decide which transmission policy is best suited to the expected message flow in the environment; *scalable topology* to choose the scheme for information sharing and communications; *object complexity* to weigh in the network performance cost of broadcasting object changes [BZWM97]; *environment persistence* to decide whether users' interactions with the environment have temporal or permanent effects in the CVRE system [LJD97]; and *user interaction* to include the user interface features desirable for a CVRE. In subsections 3.3 and 3.4 we describe how these categories allow the designer to specify the most suitable feature set for creating a collaboration framework.

2.2. User Interaction Models in Collaborative Environments

The Model-View-Controller paradigm (MVC) [KT88] is the classical user interaction model for designing user interfaces. The approach classifies all application objects in three

Table 1: *Characterization of Collaboration Features in CVREs*

Network Transmission	**Distribution:** Broadcast, multicast or unicast packages. **Latency:** Considering traffic delays and other perturbations. **Reliability:** Use of positive and negative acknowledgements. **Bandwidth:** As much as possible
Scalable Topology	**Homogeneous replication using broadcast:** Each client maintains a complete replica of the shared environment. Messages across the network transmit state information. No central control; a new client has to wait some time to accumulate enough information about the other clients. **Shared-centralized on a server:** Classical client/server model. Shared session information resides at the server. When the server fails it brings down all the clients. **Shared-distributed using small client/server groups:** Several groups of servers and clients. Uses same scheme as mobile phones cells, in which clients are connected to the adequate server. **Shared-distributed using peer-to-peer actualization:** Peer-to-peer connections among all participants, either directly or using a third party relay (broker). Changes are atomically broadcasted to all participants. It comes in two flavors: **P2Pr:** Same replicated scene graph at each node with objects stored locally. Synchronization by using callbacks, and local managing of session persistence. **P2Ps:** Sharing objects across the network in a distributed scene graph. Thin replicas shadow remote objects. Network monitors maintain correspondence within the shared space.
Object Complexity	**Light objects:** Short messages containing state, event and control information, require low latency, high-speed transmission rates. (e.g. trackers, sensors, events and status information). **Remote references:** External references shadowing remotely located objects. **Heavy objects:** Medium-atomic data. Big objects requiring reliable transmissions, but small enough to reside in the client memory, (e.g. object 3D geometry, avatars or cameras). **Real-time streams:** Large-segmented data. Data so big it has to be transmitted in pieces and/or continuously, (e.g. big geometric objects, volume information, textures, video, audio, etc.).
Environment Persistence	**Participating persistence:** The CVRE exists only while the participants are in it, and resets when all participants leave the environment. **Status persistence:** The CVRE status is stored elsewhere, to be able to use it at a later time (journaling). Allows the recording of 3D annotations to guide other clients. **Continuous persistence:** The CVRE is always active. A simulation may change the scene and its objects even if no clients are connected.
User Interaction	*Interaction strategies common to all CVEs:* –Adequate interfaces for collaborative manipulation and visualization –Teleconference capabilities (streaming video and audio) –Flexible support for data construction –Synchronous and asynchronous collaboration –Adaptive multi-resolution for less sophisticated devices –Standards and interoperability with heterogeneous systems –Replicated or shared spaces *Interaction strategies specific of CVREs:* –Use of action indicators and annotations for notification of remote events –Multiple users having several views –Use of avatars for remote user representation –Design for low latency response times

excluding categories according to user interface roles, in a process called *factoring*:

Algorithmic (semantic) objects are held in the *Model* layer.
Visual (display) objects are placed in the *View* layer.
Widget (user interface) objects translate all user manipulations into commands in the *Controller* layer.

The other common user interface paradigm is the Abstraction-Link-View (ALV) [Hil92], in which application objects are factored into abstraction, view and link layers:

Abstraction objects are models shared by all users.
View objects handle user interaction and rendering
Link objects are constraint sets that synchronize abstraction and view objects, using thin local references (*shadows*) linked to a remote object in a central repository.

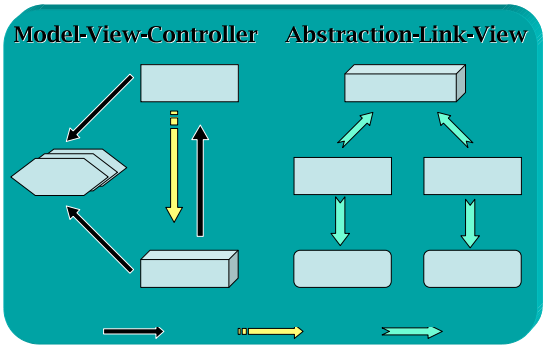


Figure 2: *MVC and ALV User Interaction Models*

A comparison of the two is shown in Figure 2. The ALV's Abstraction layer is equivalent to the MVC's Model layer, while the ALV's View component merges the View and Controller layers of MVC. The ALV's constraints connect abstractions and their views, while the MVC's Controller layer handles communications among all its objects.

In subsection 3.1 we describe a hybrid model for session management which is more suitable in networked virtual reality environments, having clients demanding reliable and continuous object flows, and needing to redisplay many complex scene changes at high speed rates.

3. The Collaboration Framework

Many virtual reality applications usually start out as helping environments for scene and object visualization, by means of special navigation and manipulation metaphors in the user interface, and using display devices ranging from simple monitors, HMDs, to inmmersive stereo projection systems such as the CAVE. Most science disciplines (and the entertainment industry as well) use VR techniques to show and enhance user experiences. As research shows, users *always* desire to share virtual experiences, either by passively showing models to prospective audiences, or by actually having active remote user participation in the environment.

Evolving collaboration out of this need usually entails the redesign and development of a (new) distributed application, developing a networking infrastructure under the visualization environments, and fraught with the usual software porting problems. Issues such as synchrony overheads, network delays, concurrent user load, and system lag degrade interaction and must be dealt without adversely graphics affecting performance.

The rationale behind our approach is that the object-oriented nature of stand-alone VR applications, usually having rendering and user-interface components, would facilitate their transformation into complete CVRE's, by allowing the seamless attachment of a network-based, collaboration-enabling component.

In the following subsections we describe the evolution of collaborative features in the proposed framework. Given that the different VR tools are spread across platforms and support varied output display systems, the ideal solution should not compromise current designs or imply extensive recoding of components when fitting the collaborative framework. Massive or large-scale implementations were discarded due to performance considerations for user administration, although the proposed framework scales well for a reasonable number (less than twenty) of participants. Based on the features described in section 2, our solution involves the implementation of a ***d**istributed user interface model*, ***m**ulti-threaded software components*, *session awareness capabilities*, and a ***n**etworking architecture*.

3.1. Distributed User Interface Model

To recast existing VR navigators as enhanced collaborative virtual reality environments, it is necessary to implement the maximal CSCW feature set allowed by the existing architecture's design, under several practical implementation restrictions. The original stand-alone behavior must remain the same, so the problem is how to obtain the maximum collaborative feature set incurring the least implementation cost. In a typical application for CSCW, the standard MVC approach proves insufficient because it does not provide for a Session layer to hold shared state and persistence properties of the remote interaction among users. The Link layer in the ALV model does provide a method to keep track of object and session changes, but it is heavily slanted toward a client-server distributed model.

We define a hybrid Model-View-Controller-Session (MVCS) approach by latching the ALV's ***Link*** concept to network pipelines connecting MVC objects, in which:

- MVC objects may not reside all together at the same network node. In a shared environment, objects may have their Model (structure and behavior) defined at one client, many different Views (renderings, at least one for each client), and control flow effected by all. For example, each

node may have multiple visualization layers (cameras), allowing for many perspectives and resolutions of the same scene.
- Since it does not matter whether objects are globally shared or locally replicated, it allows indistinctly client-server or peer-to peer approaches. Controllers operate indistinctly on both using a callback mechanism, routing to the corresponding network nodes for non-local objects, as shown on Figure 3.

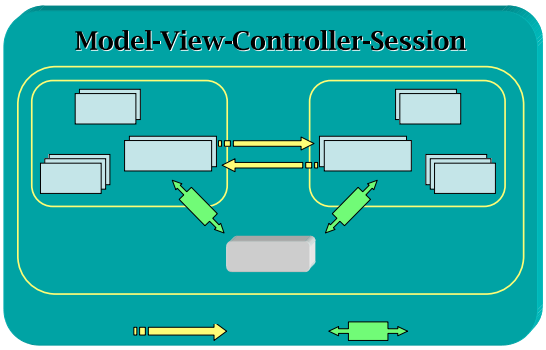


Figure 3: *Model-View-Controller-Session (MVCS) Objects **M**odel showing an external broker maintaining session states*

We part from the fact that a good VR navigator is the final product of a sound systems design, developed under a classical MVC paradigm, factoring application objects into at least two weakly cohesive software components, Graphics Rendering and User Interface, a standard software engineering practice in Computer Graphics. To develop a code framework sitting on top of VR applications, we pick from each category of Table 1 those items that better support awareness and collaboration under the MVCS reference model, transforming current navigators into certifiable distributed collaborative components.

3.2. Multithreaded software components

To decouple the Graphics Rendering and User Interface components even more, these components are wrapped as POSIX-compliant threads from a thread-pool class. Since networking capabilities must be incorporated, it is best to have them residing in its own concurrent thread. In this way, we take advantage of the underlying operating system's context switching and load a new networking component without touching functioning code. The approach is open, meaning that more concurrent threads with additional components may be created, for example, to add data acquisition from trackers or haptic devices.

A snapshot of an instantiated framework under the MVCS reference model is shown in Figure 4, detailing each software component. The Network component thread handles all communications and message parsing; the top Shared

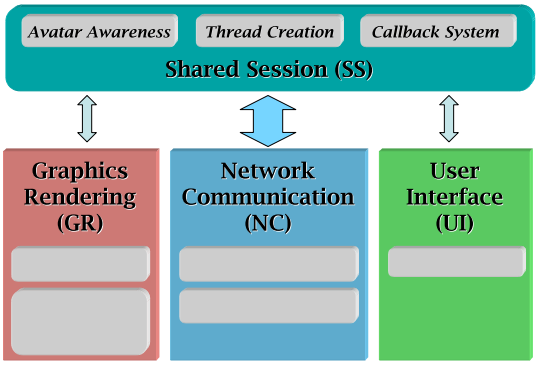


Figure 4: *Collaboration-enabling Threaded Processes. The framework includes the original components (GR and UI), and adds the session layer (SS) with the new network component (NC)*

Session Management layer launches all concurrent threads, tracks remote clients' avatars, propagate state changes to the User Interface and Graphics Rendering components by means of a callback system and is generally responsible for the emerging collaborative behavior; the Graphics Rendering and User Interface components are mostly untouched except for the connecting "glue" to the Shared Session management layer.

We implement this setup by means of an abstract class wrapper to incorporate network awareness and a corresponding message protocol. An appropriate set of mutexes has been added to avoid state inconsistencies and race conditions when updating shared information.

3.3. Session awareness capabilities

The first order of business is choosing what minimal features set is desirable for CVRE applications. We decided to include the following:

- Session administration with differentiated user roles
- Client awareness using *avatars*
- Shared annotation and 3D marker highlighting

For a client in this scenario, there must be perceptual evidence that other entities (human or otherwise) are participating in the interaction, so 3D client embodiments within the environment (avatars) are used to dynamically reflect their position and state in the scene. Clients may want to call up attention to others by special 3D signals, leaving trails and modifying the environment for other users. Some users will browse the scene, while others may have privileges for modifying scene and object properties. A distributed user interface model allows the remote manipulation of objects, and some session administration capabilities allow the environment to have memory of the interaction from a hierarchy of users.

Session administration with differentiated session roles
A manageable hierarchy of workstation roles *[stand-alone, **p**eer, incognito, slave, master]* is defined within the environment, leaving open the possibility of adding more.

A *stand-alone* **c**lient is not aware of other clients. It defaults to the original isolated behavior of the application. *Peers* are clients that communicate among themselves using the common message protocol. Users traveling ***i**ncognito* may observe scene interaction in "voyeur" mode without other clients knowing it. A *slave* is a peer that is bound to another, correspondly called a ***m**aster*, in the sense that the *master*'s current state is continuously replicated by the remote ***s**lave(s)*. These client roles are voluntary and changeable during a session.

Client awareness using *avatars*
Each client has its own 3D graphical representation, that moves about in the environment, and which may have multiple active camera perspectives at any time. Each camera may broadcast a number of state attributes, such as ***p**osition*, *orientation* **c**amera vectors, and even actual *velocity* vectors for dead reckoning calculations.

Shared annotation and 3D marker highlighting
In CSCW systems, users must not only be aware of each other, they must be able to call the attention of remote participants to some feature object of the environment. This is accomplish by pinning temporal 3D markers, graphical objects that a "guide" pins at some feature locations, such as a arrow, a message billboard, or a banner.

3.4. Networking architecture

Since communication is what enables collaboration, new software components must add network communication capabilities. We chose implementing a peer-to-peer scalable topology, the most adequate for an environment of equal participants that does not rely on a central server. Trying to fit a client/server model would have implied the creation of the server from scratch and compromises the applications' stand-alone behavior.

P2Pr [***P**eer-to-Peer with replication*] implementations and P2Ps [*Peer-to-Peer with sharing*] are equally possible in the framework. In P2Pr, each client will have its own local replica of the scene. Since only a few scene objects are modified in the session, collaboration starts as soon as all clients have loaded their common model, and situated themselves within it. A P2Ps implementation must first build a shared scene graph, to whom each individual client may incorporate whole scene chunks.

Shared nodes at the scene graph are labeled local or remote whether the object physically resides in a particular client or is a ***s**hadow* reference to a *behavior* to be fetched elsewhere. The client subscribes to a scene, objects descriptions are relayed to and from, and then rendered. For complex scenes it may take a while for the scene to load.

Having no central server, this approach requires a third party providing locating service in which clients find each other to establish joint sessions.

Thin broker for session administration
In our scheme, a third party must act the part of a *message broker*, which keep track of session participation and interaction, as seen in Figure 5. It is [loosely] based on the CORBA name service, but without IDL overhead and less services. Its internet address is known to all potential clients.

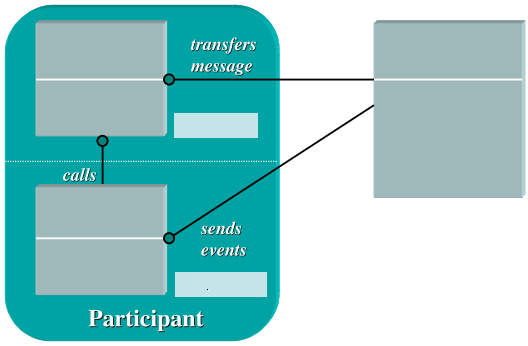


Figure 5: *Peer-to-peer Broker Class Model. Each Peer has local thin instances of other remote clients (proxies). The* broker ***k**eeps state information about session participants.*

The *broker* **i**s a thin node providing several services:

- A name service for location and client registration.
- A session management service.
- Session and client state reporting and mirroring.

Since the broker is not a bridge, client-to-client communication is the sole responsibility of the clients. To keep track of participants, all clients periodically tell the broker their current state, and the broker broadcasts to all clients the list of participants currently in the session.

Network substrate
A cross-platform networking class allows either datagram-oriented (UDP) or connection-oriented (TCP) communications under IPv4 and IPv6 networks. The NC thread, under a common message protocol implements the following basic kind of services, each one running on its own separate listening socket:

- Shared event pipeline for sending environment state changes and callback messages
- A push client for the continuous streaming of certain client properties, such as camera position and orientation
- A notifying service for the ***B**roker*.

When a client initially reports to the broker, it posts its address and the network ports on which it is listening. This is a configurable setup, to account for external firewalling rules on opening and closing specific ports, and that allows several clients to run concurrently on the same machine by choosing a different set of port numbers for each process, which

is handy way to test system performance with higher client loads than permitted by available physical workstations.

System synchronization
The framework avoids keeping a central time server by keeping *relative* time differences for every peer-to-peer connection at the client's side. The relative event local time (*timestamps*) is included in each network message. That way, clients at the other end have the freedom to choose processing incoming messages as either:

Immediate: message are processed at once.
Buffered: message are queued by timestamp.

In the first approach variable network latency may produce jumpy updates and short temporal inconsistencies. The second is more suitable for replaying events in exact time sequence, at the expense of a bigger time delay.

External real-time verbal communication channel
Most collaborative environments use at least one real-time communication channel to allow the human users behind the workstations to exchange textual/audio/video impressions about the virtual experience. The framework does not provide this service, so it is suggested using external suitable cross-platform alternatives such as Gaim, Gnomeeting and others.

4. The ALICE Virtual Reality Navigator

ALICE VR Real Time Inspector and Navigator is a standalone VR software platform for the real time inspection and navigation of very complex virtual models, developed at the Universitat Politècnica de Catalunya. It has been used in a number of applications such as navigation in urban environments or interior ship design among others.

In order to allow the users of these applications to be able to navigate and inspect complex 3D models in several VR systems, ALICE offers the following features:

- Stereoscopic visualization: either active stereo, for systems like Head Mounted Displays [Sut68], or passive stereo, for other low cost VR systems [AFB02a].
- User position and orientation tracking: allowing user's implicit interaction by following his movements and making him feel he is inspecting a real object instead of a virtual one.
- Use of varied interaction devices: being able to follow orders from a mouse, a joystick or a VR glove, for example.
- Different VR modes of execution: able to work over different VR display systems like stereoscopic tables, a CAVE, a Head Mounted Display, etc.

Apart from these external features, ALICE implements internally an extensible callback system and also many advanced algorithms in computer graphics in order to be able to work interactively with very complex scenes. It uses internally a hierarchical structure of the objects in the scene, keeping also for each one other non-geometrical information, allowing, for example, the use of textures. Among these advanced algorithms are the following:

Simplification techniques: ALICE maintains different levels of detail for the objects in the scene [ABA02].
Visibility culling: Eliminates from the visualization process those parts of the geometric model not visible from the current observer's viewpoint [ASVNB00].
Collision detection: Needed to select objects with VR gloves or haptic devices, by virtually "touching" the objects [FNB01].

4.1. Framework validation in ALICE

ALICE is an application already factored into two software components, Graphics Rendering and User Interface. The User Interface component is provided by v3.x of Qt, an object-oriented user interface toolkit under the MVC paradigm, with cross-platform deployability in MS Windows, Linux, several flavors of UNIX and MacOS X.

The decoupled callback hook system in ALICE connects user events to graphic rendering' actions by means of a indexed command list. Each element of the lists stores a settable reference (the "hook") to some other object's method (the "callback"). When an User Interface event cause triggers a particular command, its corresponding callback hook is executed with the provided event information and current environment state.

Given all the above, it was considered a suitable candidate for enhancing his collaboration features. Just changing some flags in the compilation process allows the UI component to run in its own thread if needed. The following steps were taken to fit ALICE into the framework:

1. Wrapping the GR, UI and the new NC as component attributes of the SS (session) class, and launching them in separate threads
2. Selecting a P2Pr sharing topology.
3. Selecting the messaging protocol
4. Instantiating the message parser class to process event messages, callable from the networking thread
5. Translating the processed network into local method calls, using the callback hooks mechanism already provided by ALICE
6. Adding one method call in the *user interface component's* main method to launch the networking thread at startup
7. Adding one method call in the *rendering component's* main method to sync the state of network peers just before rendering

Initially we have chosen a P2Pr scalable topology, since it was easier to implement and did not change ALICE's present stand-alone behavior, in which all clients already have local identical copies of the scene. In shared mode, the broker

indicates the name of the scene to be shared, so hopefully everyone will be placed in the same scene.

The message protocol is quite simple. There are three kinds of messages: *session*, *location* and *manipulation*. Session messages are the ones exchanged between the broker and the clients, connecting and disconnecting, internet addresses and open ports, number of active cameras, avatar appearance, global scene file, and other relevant data. Location messages are mostly for camera coordinates broadcasting among all participants. Manipulation messages (such as local client touching, grabbing, adding or modifying an object) are sent to all remote users to be processed by their callback system to maintain scene coherence among all participants. Out of the growing callback set of ALICE (around 100), only a subset of 14 affected model integrity, shared scene state and object appearance, but more may be defined in the protocol.

Since all of this happens in the new NC thread, mutexes are activated when this thread is modifying data, such as updating clients' coordinates. There was a corresponding set of mutexes placed just before rendering to avoid the race conditions common to concurrent programming.

A session is initiated by at least two participants willing to subscribe to a session, shown in Figure 5. Each client may choose a session role (usually as a ***peer***) and an avatar representation (from a menu), and keeps a list of current active interactions with other users. As they navigate, clients can chat to each other, or place 3D markers on the scene to call attention to some feature.

Clients can also take the role of "voluntary slaves" for some other user, which now becomes a camera server. The *slave* shuts down its own cameras and reflects the *master*'s camera viewpoint and actions, the latter effectively taking possession of the slave's remote display devices. This feature may also be used to "teleport" a participant to the position of another, which is very useful to avoid losing virtual eye-contact among peers. Since each node does independent renderings, it allows a client to show a wireframe representation, while another does a full render of the same scene.

5. Results

We have tested ALICE's remote collaboration and navigation services in several VR systems in our lab, and also in sessions with the Girona university (located 100 Km. from Barcelona) through a 10Mb wide area network connection. In our lab we use HMDs, the stereoscopic table [AFB*02b], the portable system [FBT04] and flat monitors (without stereo capabilities); and a similar setup at Girona university. Network traffic is generated only when an avatar changes position or orientation, or when a callbak is produced (for example when selecting an object), so no unnecessary traffic is produced.

The results obtained from our tests can be seen in the following table. The scene used on these tests (the interior of a ship) contains 20.000 polygons, but on purpose does not have complex textures because it could hinder or slow graphics performance. The table shows the results obtained in the communication of 2, 4 and 8 workstations from both labs (Girona's and our's).

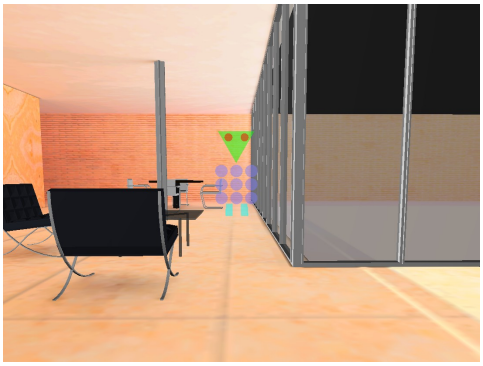
Participants	1	2	4	8
Av. Number of messages	–	2539	8067	14331
Av. Total Net. Time (msec)	–	35	31	46
Av. Roundtrip Time (msec)	–	13	46	57
Framerate	47.3	45.2	44.2	42.7

In the table we observe the average total number of messages sent through the network in a series of repeated navigation trials, each test lasting 4 minutes. The total network time (in milliseconds) gives information about how much time Alice spent in the transmission of messages during these 4 minutes tests (this means that only around 0.1-0.2% of total time was spent in network communications). The roundtrip time is also indicated in milliseconds. Since our framework uses unicast addresses, roundtrip time increases as more peers participate in the session. Finally, the table shows the average rendering framerate achieved for each case, which indicates that increasing the number of nodes affects graphics performance very slightly compared to the stand-alone performance.

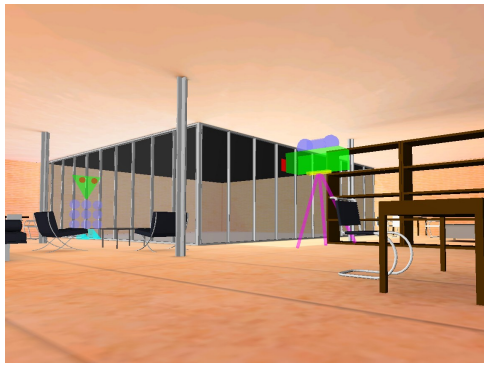
As already stated in subsection 4.1, the migration of Alice to a CVRE navigator was very easy. Based on the fact that the application was already designed considering the rendering and the user interface as separated components, its port to use our framework only required to define a message protocol, connect the appropriate callback hooks, and add two method calls in its code in order to attach the application to the new network and session parts. Following the same migration scheme, it would be easy as well to transform any other VR application into a collaborative VR application. In fact we are presently porting another application built in our lab which addresses inspection and management of medical models.

Although we have not done experiments in slow networks, we simulated a fictitious one and we found out that sequential processing of arriving avatar information may cause clients to fall out-of-sync. In order to minimize these latency problems, there is an option to process only the most recently received information packet from each camera in the environment, at the expense of a somewhat jumpier navigation.

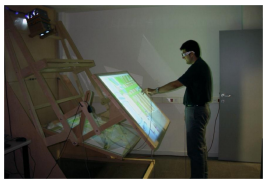
The proposed mechanisms for camera management and sharing is reasonably easy to learn for users and seem to be adequate for collaboration tasks. We want to make some experiments with untrained users soon in order to have a more accurate perception of ease-of-use.



(a) "camera" avatar's viewscene



(b) An *incognito* avatar's viewscene, showing the "camera" and "upecito" interact



(c) Participant at Stereoscopic table

(d) Participant with HMD

Figure 6: *Testing collaboration between* peers*. Two clients (embodied by the "camera" and "upecito" avatars) observe each other while navigating the environment*

6. Conclusions and Future Work

Based on a characterization of generic collaborative features for VR systems, we have proposed a versatile framework for evolving collaborative capabilities in stand-alone VR navigators. Our approach incorporates a hybrid distributed user interaction model, multi-threaded software components, network communications under a peer-to-peer scalable topology, message passing channels with a custom protocol, and changeable user roles in a multicamera subscription model.

The framework's development has been validated by a fast porting of the ALICE VR Navigator. The generic cross-platform design allows an easy migration of similar VR applications into complete collaborative virtual reality environments. As for future work, we are working on extending the collaborative breadth of the framework by including in the Session layer a fourth thread for handling haptic devices, adding high frequency force-feedback events to the interactive session repertoire. Given the huge scene size of current VR scenes and objects, we plan to migrate applications towards a peer-to-peer with sharing scheme (P2Ps), and also to allow the incremental streaming of multiresolution objects to improve rendering performance and scalability.

References

[ABA02] ANDÚJAR C., BRUNET P., AYALA D.: Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graphics 20*, 6 (2002).

[AFB02a] ANDÚJAR C., FAIRÉN M., BRUNET P.: Affordable projection system for 3d interaction. In *1st Ibero-American Symposium in Computer Graphics* (July 2002), University of Minho, Portugal.

[AFB*02b] ANDÚJAR C., FAIRÉN M., BRUNET P., CEBOLLADA V., TRUEBA R., VALOR D.: Sistema de proyección inmersivo para la interacción con modelos 3d. In *Actas del tercer congreso Interacción Persona–Ordenador* (Junio 2002), Universidad Carlos III, Madrid.

[ASVNB00] ANDÚJAR C., SAONA-VÁZQUEZ C., NAVAZO I., BRUNET P.: Integrating occlusion culling and levels of detail through hardly-visible sets. *Computer Graphics Forum 19*, 3 (Aug. 2000).

[BZWM97] BRUTZMAN D., ZYDA M., WATSEN K., MACEDONIA M.: Virtual reality transfer protocol (vrtp) design rationale. In *Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE): Sharing a Distributed Virtual Reality* (Massachusetts Institute of Technology, Cambridge Massachusetts, June 1997).

[CH93] CARLSSON C., HAGSAND O.: Dive: A platform for multi-user virtual environments. *Computers and Graphics 17*, 6 (1993), 663–669. http://www.sics.se/dce/dive.html.

[CMBZ00] CAPPS M., MCGREGOR D., BRUTZMAN D., ZYDA M.: Npsnet-v: A new beginning for dynamically extensible virtual environments. *IEEE Computer Graphics and Applications 20*, 5 (2000), 12–15.

[CSS99] C. S., SCHUMMER J., SEITZ P.: Modeling collaboration using shared objects. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work* (Phoenix, Arizona, United States, 1999).

[DGC*98] DUCE D., GIORGETTI D., COOPER C., GALLOP J., JOHNSON K., SEELIG C.: Reference models for distributed cooperative visualization. *Computer Graphics Forum 17*, 4 (1998), 219–233.

[DKS*99] DERIGGI F., KUBO M., SEMENTILLE A., FERREIRA J., DOS SANTOS S., KIRNER. C.: Corba platform as support for distributed virtual environments. In *Proceedings of IEEE, Virtual Reality'99* (Houston, Texas, March 1999).

[FBT04] FAIRÉN M., BRUNET P., TECHMANN T.: Minivr: A portable virtual reality system. *Computers & Graphics (to appear) 28*, 2 (April 2004).

[FNB01] FRANQUESA-NIUBO M., BRUNET P.: *Collision Detection using Multiresolution Kdtrees*. Tech. Rep. LSI-01-36-R, UPC, 2001.

[GB94] GREENHALGH C., BENFORD S.: Massive, a collaborative virtual environment for teleconferencing. *ACM Transactions on Computer, Human Interaction. 2*, 3 (1994), 239–261.

[GFPB02] GREENHALG C., FLINTHAM M., PURBRICC J., BENFORD S.: Application of temporal links: Recording and replaying virtual environments. In *Proceedings of the IEEE VR, 2002* (2002).

[Hil92] HILL R. D.: The abstraction-link-view paradigm: Using constraints to connect user interfaces to applications. In *Proceedings of the SIGCHI conference on Conference on Human Factors and Computing Systems* (Monterey, California, United States, 1992), pp. 335–342.

[HJCN01] HARTING P., JUST C., CRUZ-NEIRA C.: Distributed virtual reality using octopus. In *Proceedings of IEEE Virtual Reality 2001* (March 2001), pp. 53–62.

[HSFP99] HESINA G., SCHMALSTIEG D., FUHRMANN A., PURGATHOFER W.: Distributed open inventor: A practical approach to distributed 3d graphics. In *Proceedings of ACM VRST'99* (1999), pp. 74–80.

[KSA*03] KELSO J., SATTERFIELD S. G., ARSENAULT L. E., KETCHAN P. M., KRIZ R. D.: Diverse: a framework for building extensible and reconfigurable device-independent virtual environments and distributed asynchronous simulations. *Presence: Teleoper. Virtual Environ. 12*, 1 (2003), 19–36.

[KT88] KRASNER G. E., T. S.: Pope, a cookbook for using the model-view controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming 1*, 3 (1988), 26–49.

[LJD96] LEIGH J., JOHNSON A. E., DEFANTI T.: Cavern: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. *Journal of Virtual Reality Research, Development and Applications 2*, 2 (1996), 217–237.

[LJD97] LEIGH J., JOHNSON A. E., DEFANTI T. A.: Issues in the design of a flexible distributed architecture for supporting persistence and interoperability in collaborative virtual environments. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM)* (1997), ACM Press, pp. 1–14.

[Mau00] MAUVE M.: How to keep a dead man from shooting. In *Lecture Notes in Computer Science* (2000), vol. 1905, Springer-Verlag Heidelberg.

[MF98] MACINTYRE B., FEINER S.: A distributed 3d graphics library. In *COMPUTER GRAPHICS (Proceedings of ACM SIGGRAPH 98)* (July 1998), pp. 361–370.

[MW01] MEISTER M., WÜTHRICH C. A.: On synchronized simulation in a distributed virtual environment. In *WSCG 2001 Conference Proceedings* (2001), Skala V., (Ed.).

[Sty96] STYTZ M. R.: Distributed virtual environments. *Computer Graphics and Applications 16*, 3 (1996), 19–31.

[Sut68] SUTHERLAND I.: A head-mounted three-dimensional display. In *AFIPS Conference Proceedings* (1968), vol. 33, pp. 757–764.

[TFN03] THEOKTISTO V., FAIRÉN M., NAVAZO I.: Collaborative virtual reality in the alice platform. In *Proceedings of spanish conference of Computer Graphcis (CEIG 2003)* (2003), pp. 261–276.

[Tra99] TRAMBEREND H.: Avocado: A distributed virtual reality framework. In *Proceedings of the IEEE Virtual Reality* (1999).

[WLG99] WEDLAKE M., LI K. F., GUIBALY F. E.: The NAVL distributed virtual reality system. *Lecture Notes in Computer Science 1554* (1999), 177–193.

[ZHC*00] ZELEZNIK B., HOLDEN L., CAPPS M., ABRAMS H., MILLER T.: Scene-graph-as-bus: Collaboration

between heterogeneous stand-alone 3-d graphical ap-**p**lications. In ***E****URO*GRAPHICS *2000* (2000), Gross **M**., Editors) F. H. G., (Eds.), vol. 19.

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports - 2004

- LSI-04-1-R : *Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations*, Rodríguez, E. and Kapur, D.
- LSI-04-2-R : *Comparison of Methods to Predict Ozone Concentration* , Orozco, J.
- LSI-04-3-R : *Towards the definition of a taxonomy for the cots product´s market* , Ayala, Claudia P.
- LSI-04-4-R : *Modelling Coalition Formation over Time for Iterative Coalition Games*, Mérida-Campos, C. and Willmott, S.
- LSI-04-5-R : *Illegal Agents? Creating Wholly Independent Autonomous Entities in Online Worlds*, Willmott, S.
- LSI-04-6-R : *An Analysis Pattern for Electronic Marketplaces*, Queralt, A. and Teniente, E.
- LSI-04-7-R : *Exploring Dopamine-Mediated Reward Processing through the Analysis of EEG-Measured Gamma-Band Brain Oscillations*, Vellido, A. and El-Deredy, W.
- LSI-04-8-R : *Studying Embedded Human EEG Dynamics Using Generative Topographic Mapping*, Vellido, A. and El-Deredy, W. and Lisboa, P.J.G.
- LSI-04-9-R : *Similarity and Dissimilarity Concepts in Machine Learning*, Orozco, J.
- LSI-04-10-R : *A Framework for the Definition of Metrics for Actor-Dependency Models*, Quer, C. and Grau, G. and Franch, X.
- LSI-04-11-R : *QM: A Tool for Building Software Quality Models*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-12-R : *COSTUME: A Method for Building Quality Models for Composite COTS-based Software Systems*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-13-R : *Enabling Collaboration in Virtual Reality Navigators*, Theoktisto, V. and Fairén, M. and Navazo, I.

Hardcopies of reports can be ordered from:

Núria Sanchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Campus Nord, Mòdul C6
Jordi Girona Salgado, 1-3
03034 Barcelona,Spain
nurias@lsi.upc.es

See also the Departament WWW pages, http://www.lsi.upc.es/